Amendments to the Specification:

[0039] In another embodiment, the abstract representation 14 of test cases 12 is enriched with information from an application metadata repository 54. The abstract representation 14 of test cases 12 can be enriched by extracting values for those attributes 26 of application objects 24 associated with the test cases 12 that are missing in the incoming test scripts. The enrichment of test cases 12 can decouple test cases 12 and their recording or authoring environments, and the like, and allow usage of attributes 26 that are stable within an application metadata representation 54. For example, an internal identification field 56 within the application metadata repository 54 can be utilized to identify a given object 24 instead of a language dependent display label. This improves the reusability of the test case 12. Because different test execution environments can use different attributes 26 to identify the same application object [[16]] 24, such decoupling provides platform independence.

[0042] Logic 122 is provided to implement semantic analysis and convert AST 120 to an abstract internal test case representation [[122]] based on an Application Object Model (AOM). Logic 124 enriches the abstract internal test case representation with information from an application metadata repository [[126]]. Logic 126 separates application object attributes and input data from external interaction sequencing to provide automatic parameterization.

~~**[0043]** Referring to Figures 1 and 2, one embodiment of the present invention is a system 10 and method for transforming test cases. Test cases 12 are imported that are written in one or more scripting languages. Test cases 12 are then converted to an abstract representation 14 which includes one or more application states 16, external interaction sequences 18 and input data 20. Abstract representations 14 are stored in a database system 22. A variety of different database systems 22 can be utilized including but not limited to, a relational database management system, an XML database management system, and the like.~~

~~**[0044]** An application state 16 represents a runtime snapshot of an application under test which defines the context of external interaction. In one embodiment, illustrated in Figure 3, application state 16 is a set of application objects 24, such as a~~

~~web page or a window control or an account object, for example. Each application object 24 is associated with a set of attributes 26 and their values. For example, a web page can have an attribute 26, called url, which contains the Uniform Resource Locator (URL) corresponding to the current web page, and an attribute 26, called title, which contains the title of the current web page. In one embodiment, the set of applications states 16 is represented in the test case 12 and are arranged in a hierarchical manner.~~

~~**[0045]** Scripting languages utilized can be typed or untyped programming languages used for recording or authoring test cases. External interaction sequences 18 can represent events invoked by external agents 28 on application objects 24. External agents 28 can be either human agents or other software agents. Interaction sequencing can include flow control structures 32 for capturing sequential, concurrent, looping, conditional interactions, and the like.~~

~~**[0046]** As shown in Figure 4, in one embodiment, a syntax analysis 34 can be implemented for incoming scripts. Syntax analyzer 34 can be implemented one for each scripting language. Syntax analyzer 34 can utilize rules of syntax analysis 36 that are specified in Extended Backus-Naur Form (EBNF). Syntax analysis can generate a parse tree in the form of an Abstract Syntax Tree (AST) 38. One embodiment of a method of handling scripts with the present invention is illustrated in the Figure 5 flowchart.~~

~~**[0047]** In one embodiment, a semantic analysis 40 is implemented that converts the AST 38 to an abstract test case representation 42 based on an Application Object Model (AOM) 44. Semantic analysis 40 decomposes the test cases represented as an AST 38 into application state 16, external interaction sequences and input data.~~

~~**[0048]** As illustrated in Figure 6, AOM 44 can be a metadata representation for modeling application under test. Components of the metadata representation include, but are not limited to, application object type definitions 48 for application objects 24, attribute definitions 50 for each application object 24 type, definitions of methods and events 52 that are supported by each application object 24 type, definitions of effects of events 52 on an application state 16, and the like. One embodiment of a method of performing semantic analysis with the present invention is illustrated in the Figure 7 flowchart.~~

[0049] Application object type definitions 48 can include additional categorization of each application object 24 type, and can be, (i) hierarchical, (ii) container and (iii) simple. The hierarchical object types are associated with an application state 16 of its own. Application object types 16 that can contain instances of other objects are container types. For example, a web page can be represented by a hierarchical object type and table within the web page by a container type. A label in the page is represented by a simple object type. The state associated with a hierarchical application object type 16 is a modal application state or a nonmodal application state. A modal application state restricts possible interactions to application object 24 instances available within a current application state 16. A dialog window for example restricts all user interactions to the current dialog window.

[0050] The effects of events 52 on an application state 16 capture one or more consequences of an event 52 to the application state 16. A consequence of an event 52 can be, creation of an instance of an object of a given type, deletion of an instance of an object type, modification of attributes of an existing object of type, selection of an instance of an object type, and the like.

[0051] Creation or selection of a hierarchical object type can result in formation of, a new application state 16, selection of the application state 16 associated with the object type, and the like.

[0052] In another embodiment, the abstract representation 14 of test cases 12 is enriched with information from an application metadata repository 54. The abstract representation 14 of test cases 12 can be enriched by extracting values for those attributes 26 of application objects 24 associated with the test cases 12 that are missing in the incoming test scripts. The enrichment of test cases 12 can decouple test cases 12 and their recording or authoring environments, and the like, and allow usage of attributes 26 that are stable within an application metadata representation 54. For example, an identification field 56 within the application metadata repository 54 can be utilized to identify a given object 24 instead of a language dependent display label. This improves the reusability of the test case 12. Because different test execution environments can use different attributes 26 to identify the same application object 16, such decoupling provides platform independence.

[0053] In one embodiment, application object attributes 26 and input data are separated from external interaction sequencing to provide automatic parameterization. By automatically separating the data from the test case scenario, the 10 system dramatically reduces the manual labor involved to parameterize the scripts. Using the application object model, input data associated with each event 52 is separated from the scenario definition. The same process is applied to storing the object attributes 26. The input data definition forms a nested table data type definition that is driven for the events 52 involved in the scenario, and object event definitions in the application object model. This allows any data sets that match this definition to be applied to the same set of scenarios.

[0054] In another embodiment of the present invention, illustrated in Figure 8, a computer system 110 includes a processor 112 coupled to a memory 114. Memory 114 stores program instructions 116 executable by processor 112 for converting test cases to an abstract representation that includes application state, external interaction sequences and input data. A database 116 stores abstract representation of test cases. A syntax analyzer 118 can be included for incoming scripts. Syntax analyzer 118 generates a parse tree in the form of an Abstract Syntax Tree (AST) 120.

[0055] Logic 122 is provided to implement semantic analysis and convert AST 120 to an abstract test case representation 122 based on an Application Object Model (AOM). Logic 124 enriches the abstract test case representation with information from an application metadata repository 126. Logic 126 separates application object attributes and input data from external interaction sequencing to provide automatic parameterization.